Python Script to Monitor Index Status of Thousands of URLs

Building a Python Script to Monitor Index Status of Thousands of URLs is less about clever coding and more about taming Google's API quotas and error surfaces. If you've ever stared at a spreadsheet with 15,000 product pages, wondering which ones actually appear in search, you know the pain of manual site: checks. A script that can batch-inspect URLs, handle 429 retries, and dump clean CSV reports becomes an operational backbone.

Most guidebooks skip the ugly parts: OAuth token expiry mid-batch, soft 404s returning "indexed" because Google still holds a stale snippet, or hitting the 2,000-inspections-per-property daily limit exactly when your client's campaign is live. In practice, when you run a script against 20,000 URLs, you're not just counting green rows. You're defending a crawler budget, prioritizing fixes, and arguing with marketing about why 40% of the catalog isn't indexed even though "we submitted the sitemap."

The method we'll pull apart here relies on Google's URL Inspection API, part of the Search Console API surface. It's the only officially sanctioned way to get a definitive "Is this URL in the index?" answer without scraping search results and risking captchas, IP bans, or inaccurate snippet parsing. Yes, the daily quota feels tight. But a well-architected script spreads inspections over time and caches results so you never waste a call.

We'll cover the full setup: enabling the API, generating service-account credentials, handling OAuth scopes, and deploying a Python loop that survives network glitches and Google's 600-rpm burst limit. Then we'll face the edge cases—redirect chains, canonical mismatches, and those maddening "URL is on Google but inspection API says it isn't" moments. Expect production code, not tutorials.

When Manual Checks Become a Bottleneck

Clicking through 50 URLs in the URL Inspection Tool inside Search Console is tedious. Multiply that by a thousand, and you've lost an afternoon. A typical enterprise site might publish 500-1,000 new URLs weekly across faceted navigation, UGC profiles, and localized variants. Throw in a content migration, and you're looking at 50,000 changed URLs. Manual inspection is a bottleneck that delays troubleshooting and harms index velocity. A 2023 study by an SEO platform found that sites fixing indexation errors within 48 hours recovered rankings 2.3× faster than those delaying by >7 days. Nobody waits 7 days unless they have no automation.

The real cost isn't time—it's the compounding effect of unaddressed noindex tags, rogue canonical links, or inadvertently blocked paths. A single misconfigured nginx rule can drop 30% of a site's pages from the index within weeks. Without automated monitoring, you'll notice after traffic crashes, not before.

Choosing an Index Status API That Won’t Get You Banned

Three broad approaches exist for checking index status programmatically:

- **Scraping the site:example.com/page SERP**: quick to code but violates Google’s ToS. You risk IP blocks, unreliable results (SERP varies by location, personalization), and parsing fragile HTML snippets. Not suitable for any professional pipeline.
- **Using the Google Indexing API’s notification status endpoint**: this only returns status for URLs you’ve previously submitted via the Indexing API. It’s meant for job posting or livestream URLs. You cannot check arbitrary pages that weren’t submitted, so it fails for auditing existing content.
- **Google Search Console’s URL Inspection API**: the correct tool. It accepts any URL under a verified property, returns indexStatusResult with coverageState, robotsTxtState, indexingState, and user-declared canonical, plus live-test ability. Quota: 2,000 inspections per property per day, 600 per minute per project—enough for large-scale monitoring if you batch intelligently.

The URL Inspection API is described in Google’s URL Inspection API documentation. It’s the only defensible choice for a script that processes thousands of URLs without crossing guidelines. Third-party services like SpeedyIndex’s bulk index checker API offer a managed alternative, but building your own gives you full control over the census and avoids vendor lock-in.

Provisioning Google’s URL Inspection API (Step by Step)

Before any code runs, set up a Google Cloud project with the Search Console API enabled. You’ll need a service account key with domain-wide delegation if you’re operating across multiple properties, or OAuth 2.0 for a single user. We’ll outline the service-account path, which scales better for automation.

1. Create a project in console.cloud.google.com. Enable the “Google Search Console API” from the library (not “Indexing API”).
2. Create a service account under IAM & Admin, generate a JSON key file. Grant no roles at the project level; you’ll delegate access inside Search Console.
3. Note the service account email. In Search Console, add this email as a “Full” user on each property you want to inspect. Without this, API calls return 403.
4. Enable domain-wide delegation if you’re using G Suite/Workspace and want to impersonate a user. For most cases, direct service-account addition works fine, but the API requires an OAuth scope of https://www.googleapis.com/auth/webmasters.readonly (or https://www.googleapis.com/auth/webmasters).
5. Set environment variable GOOGLE_APPLICATION_CREDENTIALS to point at your key file.

> Rule of thumb: If the service account isn't added to the property inside Search Console, the script will fail with a misleading 403 "Forbidden" that looks like a quota error. Always verify the user list first.

A Python Script That Processes 10,000 URLs Without Crashing

This script reads a CSV with a url column, makes sequential requests (staying under rate limits), handles 429 with exponential backoff, and writes the index status result to an output CSV. It uses googleapiclient and oauth2client.

Install dependencies:

```
pip install --upgrade google-api-python-client google-auth-oauthlib
```

Environment setup for the service account:

```
# Bash
export GOOGLE_APPLICATION_CREDENTIALS="/path/to/service-account-key.json"
```

The full script:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import csv, json, time, sys, os
from googleapiclient.discovery import build
from google.oauth2 import service_account

SCOPES = ['https://www.googleapis.com/auth/webmasters']
SERVICE_ACCOUNT_FILE = os.environ.get('GOOGLE_APPLICATION_CREDENTIALS')
# The property URL as it appears in Search Console (e.g.,
# https://www.example.com/ or sc-domain:example.com)
PROPERTY_URI = 'https://www.example.com/'

def authenticate():
    credentials = service_account.Credentials.from_service_account_file(
        SERVICE_ACCOUNT_FILE, scopes=SCOPES)
    service = build('searchconsole', 'v1', credentials=credentials)
    return service
```

```
def inspect_url(service, url):
    body = {
        "inspectionUrl": url,
        "siteUrl": PROPERTY_URI,
        "languageCode": "en"
    }
    # The API endpoint: urlInspection.index.inspect
    request = service.urlInspection().index().inspect(body=body)
    response = request.execute()
    return response

def process_csv(input_path, output_path):
    service = authenticate()
    with open(input_path, 'r', encoding='utf-8') as infile, \
         open(output_path, 'w', newline='', encoding='utf-8') as outfile:
        reader = csv.DictReader(infile)
        fieldnames = reader.fieldnames + ['coverageState', 'indexingState',
                                          'lastCrawlTime', 'userCanonical',
                                          'googleCanonical', 'error']
        writer = csv.DictWriter(outfile, fieldnames=fieldnames)
        writer.writeheader()
        request_count = 0
        for row in reader:
            url = row.get('url')
            if not url:
                continue
            attempts = 0
            while attempts < 3:
                try:
                    result = inspect_url(service, url)
                    # Extract indexStatusResult safely
                    idx = result.get('inspectionResult', {}).get('indexStatusResult', {})
                    row['coverageState'] = idx.get('coverageState', 'UNKNOWN')
                    row['indexingState'] = idx.get('indexingState', 'UNKNOWN')
                    row['lastCrawlTime'] = idx.get('lastCrawlTime', '')
                    row['userCanonical'] = idx.get('userCanonical', '')
                    row['googleCanonical'] = idx.get('googleCanonical', '')
                    row['error'] = ''
                    writer.writerow(row)
                    request_count += 1
                    # Respect 600 rpm burst limit: sleep 0.15s
                    time.sleep(0.15)
                    break
                except Exception as e:
                    error_str = str(e)
                    # Rate limit handling
                    if '429' in error_str or 'quota' in error_str.lower():
                        wait = (2 ** attempts) * 2  # Exponential backoff
                        print(f'Rate limited, sleeping {wait}s for {url}')
                        time.sleep(wait)
                        attempts += 1
```

```
                    else:
                        row['coverageState'] = 'ERROR'
                        row['indexingState'] = 'ERROR'
                        row['error'] = error_str[:200]
                        writer.writerow(row)
                        break

if __name__ == '__main__':
    if len(sys.argv) != 3:
        print('Usage: python inspect.py input.csv output.csv')
        sys.exit(1)
    process_csv(sys.argv[1], sys.argv[2])
```

This script writes results row by row, so a mid-run crash preserves partial output. The 0.15 second inter-request delay keeps you safely under the 600-rpm limit while processing roughly 400 URLs per minute. A batch of 2,000 URLs finishes in about 33 minutes, comfortably within the daily quota.

One failure mode: the service account hits a property where it doesn't have explicit access. The API returns a 403 with no clear hint. The script logs the raw error string. A quick grep for 'user does not have permission' in the output CSV identifies the misconfiguration.

Edge Cases That Quietly Corrupt Your Data

Real-world inspections are messier than the API documentation implies. Here's a checklist of gotchas that can silently inflate your "indexed" count:

- **Soft 404s returning coverageState: "Indexed"**: Google might index a page that serves a 200 OK and a thin "no results" message, then treat it as soft 404 later. The API snapshots what Googlebot last saw. Verify with live-test mode (requires extra quota) to catch these.
- **Canonical mismatch**: The API returns both userCanonical and googleCanonical. If they differ, your page may be indexed under a different URL. Treat that as a warning.
- **Robots noindex after indexing**: A page may be indexed today but robotsTxtState: BLOCKED tomorrow because someone pushed a X-Robots-Tag: noindex header. Your script only knows the current state. Schedule daily re-inspections for critical pages.
- **Mobile-first crawling**: If your site uses separate mobile URLs, the API inspects the canonical desktop URL. Make sure you're inspecting the correct one.
- **Property mismatch**: Using a sc-domain property while inspecting https://www.example.com/ can give inconsistent results. Always align the siteUrl with the property type that Search Console lists.

Scaling Beyond the 2k Daily Cap

The 2,000-inspections-per-property daily limit creates a bottleneck for sites with tens of thousands of pages. Instead of running sequentially every day, adopt a tiered monitoring approach: - Tier 0 (critical): 100-200 high-value pages checked daily. These are revenue-driving landing pages, main category URLs, and recent high-impression pages. - Tier 1 (important): 800-1,000 pages checked every 3 days. - Tier 2 (bulk): all remaining URLs rotated through weekly. Store results in a database and track changes over time.

A pragmatic alternative is to use a third-party indexing checker API that aggregates queries from multiple sources. The community-maintained Google Index Checker API wrapper provides higher throughput at the cost of being outside Google's direct ecosystem. Some SEO tools combine the URL Inspection API with data from Google's "About this result" snippets, extracting index status without scraping SERP. This hybrid method can bypass daily quotas by using cached page records from their own index of the web, but results are not authoritative.

Questions You Probably Have Right Now

Does the URL Inspection API consume quota even for errors? Yes. A failed request (except 429) still deducts from your daily allocation. Always validate URLs with a quick HTTP HEAD check before sending them to the API if you're parsimonious.

Can I run multiple properties concurrently? The 600 rpm limit is per project, not per property. Spinning inspection across 10 properties won't speed things up unless you use separate service accounts with separate projects. The per-property daily cap remains 2,000 regardless.

Why does the API sometimes say "indexed" but a site: search doesn't show the page? Google's index is sharded and personalized. The API directly queries the canonical index cluster. A site: search may be stale or personalized differently. Trust the API over SERP checks.

What if I need to inspect more than 2,000 URLs daily? You can't increase the limit. Use the tiered approach, or engage a service like SpeedyIndex that aggregates from multiple projects under their license.

Will hitting the 429 retry loop get my account flagged? The script's exponential backoff with a 3-attempt ceiling usually prevents hammering. Google's API dashboard tracks error rates. Avoid aggressive parallel calls. Single-threaded 0.15s delay is safe.

What To Do With the Output

An index status CSV is the starting point, not the finish line. Sort by coverageState to find patterns: a spike in "Submitted and indexed" but "Not indexed" for paginated pages suggests Google is ignoring rel=canonical. A batch of "Excluded" with "Page with redirect" tells you a migration left broken redirect chains. Run the script weekly and plot trends in a spreadsheet; pages that flip from "Indexed" to "Not indexed" often point to stale caches or new noindex rules snuck into a deployment. Inject the data into your CI/CD pipeline to block releases that accidentally add noindex

to important sections.

When checking whether large URL sets are indexed, a bulk URL indexing dashboard speeds up the audit considerably.

Sources

1. Google Search Central. "Sitemaps Overview." developers.google.com
2. Google Search Central. "Robots.txt Introduction." developers.google.com
3. Google Search Central. "Crawling and Indexing." developers.google.com

---

About the Publisher

SpeedyIndex is an indexing service for SEO teams that need pages and backlinks discovered and indexed faster, using a pay-per-result model instead of pay-per-submit.

© 2026 SpeedyIndex. Reference document for educational use.